AUTOJUDGE: PROJECT REPORT

Aayushi Bhardwaj – 23112001

• Project Overview & Problem Statement

Online coding platforms like Codeforces, Kattis, and CodeChef classify programming problems as Easy, Medium, or Hard and assign numerical difficulty scores. However, this process relies on slow human judgment and user feedback.

This project develops an intelligent machine learning system that automatically predicts:

- Problem Class (Easy/Medium/Hard) using classification, and
- Problem Score (numerical difficulty value) using regression,

using only the textual content of the problem, including the problem description, input description, and output description.
The system was trained on a dataset of 4,111 Kattis problems and leverages TF-IDF text features for prediction.
Predictions are delivered through a Streamlit web application, where users can paste new problem text and instantly see the predicted difficulty class and score.

Best Results:
Classification: 52% accuracy using Random Forest
Regression: MAE = 0.743778 , RMSE = 0.920731

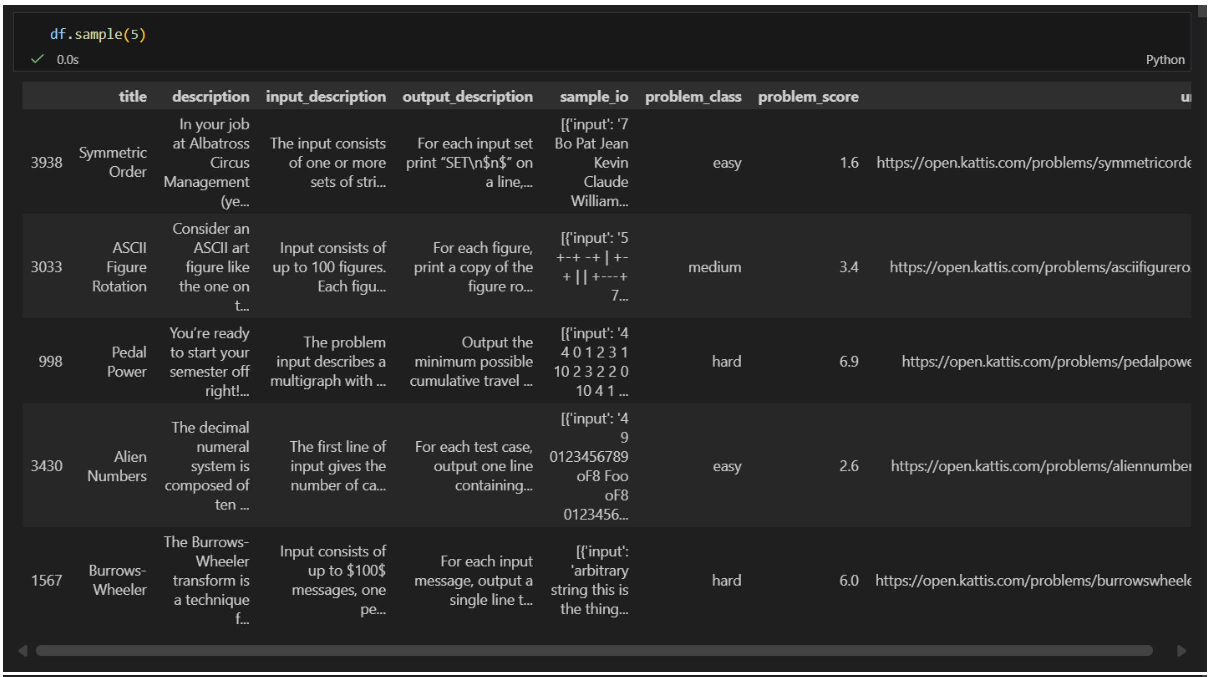
• Dataset Description:

Source: https://raw.githubusercontent.com/AREEG94FAHAD/TaskComplexityEval-24/refs/heads/main/problems_data.jsonl
Total problems: 4,111 from Open Kattis
Columns Used:

- title – Problem name
- description – Main problem statement
- input_description – Input format
- output_description – Output format
- problem_class – Easy / Medium / Hard (classification target)
- problem_score – 1.1 to 9.7 (regression target)

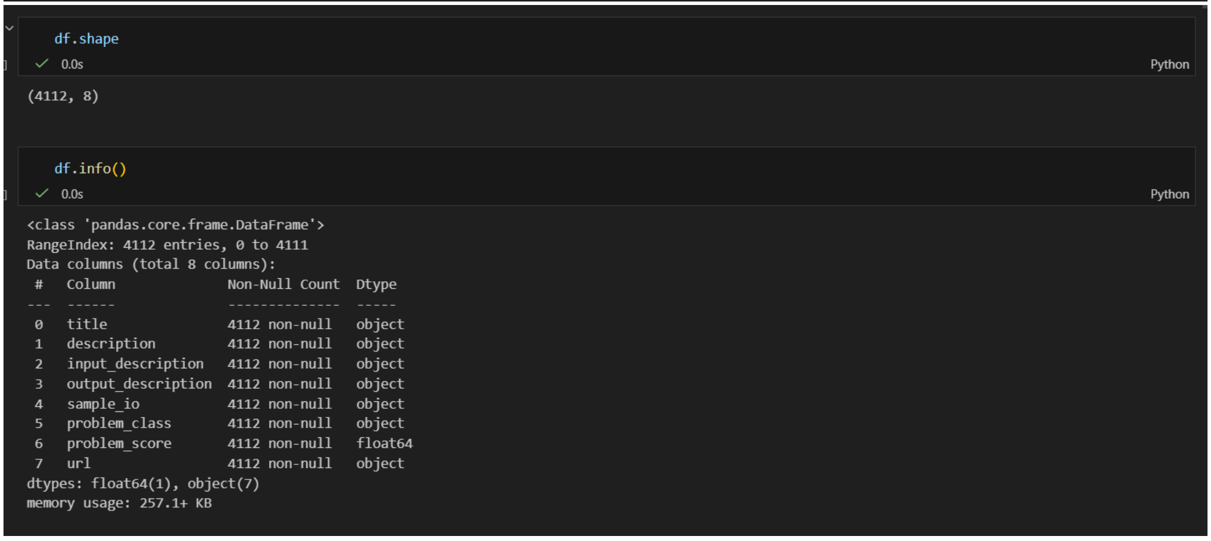
Class Distribution:

- Hard: 47% (1,941 problems)
- Medium: 34% (1,405 problems)
- Easy: 19% (766 problems)



```
df.sample(5)
```
✓ 0.0s Python

	title	description	input_description	output_description	sample_io	problem_class	problem_score	ur
3938	Symmetric Order	In your job at Albatross Circus Management (ye...	The input consists of one or more sets of stri...	For each input set print "SET\n$n$" on a line,...	[{'input': '7 Bo Pat Jean Kevin Claude William...	easy	1.6	https://open.kattis.com/problems/symmetricorde
3033	ASCII Figure Rotation	Consider an ASCII art figure like the one on t...	Input consists of up to 100 figures. Each figu...	For each figure, print a copy of the figure ro...	[{'input': '5 +-+ -+ | +- + || +---+ 7...	medium	3.4	https://open.kattis.com/problems/asciifigurero
998	Pedal Power	You're ready to start your semester off right!...	The problem input describes a multigraph with ...	Output the minimum possible cumulative travel ...	[{'input': '4 4 0 1 2 3 1 10 2 3 2 2 0 10 4 1 ...	hard	6.9	https://open.kattis.com/problems/pedalpowe
3430	Alien Numbers	The decimal numeral system is composed of ten ...	The first line of input gives the number of ca...	For each test case, output one line containing...	[{'input': '4 9 0123456789 oF8 Foo oF8 0123456...	easy	2.6	https://open.kattis.com/problems/aliennumber
1567	Burrows-Wheeler	The Burrows-Wheeler transform is a technique f...	Input consists of up to 100 messages, one pe...	For each input message, output a single line t...	[{'input': 'arbitrary string this is the thing...	hard	6.0	https://open.kattis.com/problems/burrowswheele



```
df.shape
```
✓ 0.0s Python

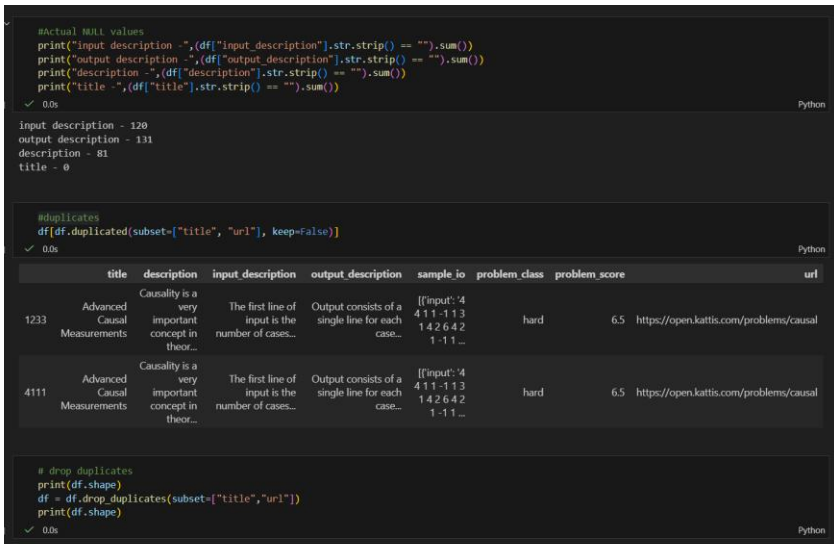
```
(4112, 8)
```

```
df.info()
```
✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4112 entries, 0 to 4111
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   title               4112 non-null   object 
 1   description         4112 non-null   object 
 2   input_description   4112 non-null   object 
 3   output_description  4112 non-null   object 
 4   sample_io           4112 non-null   object 
 5   problem_class       4112 non-null   object 
 6   problem_score       4112 non-null   float64
 7   url                 4112 non-null   object 
dtypes: float64(1), object(7)
memory usage: 257.1+ KB
```

Data Preprocessing

1. Missing Values: 120 rows had empty strings ("") in input_description and 131 in output_description, and 81 in description while title had no missing values. These are not true nulls.
2. Duplicates: 1 duplicate row was found and removed with df.drop_duplicates(), resulting in a clean dataset of 4,110 rows.



```
#Actual NULL values
print("input description -",(df["input_description"].str.strip() == "").sum())
print("output description -",(df["output_description"].str.strip() == "").sum())
print("description -",(df["description"].str.strip() == "").sum())
print("title -",(df["title"].str.strip() == "").sum())
```

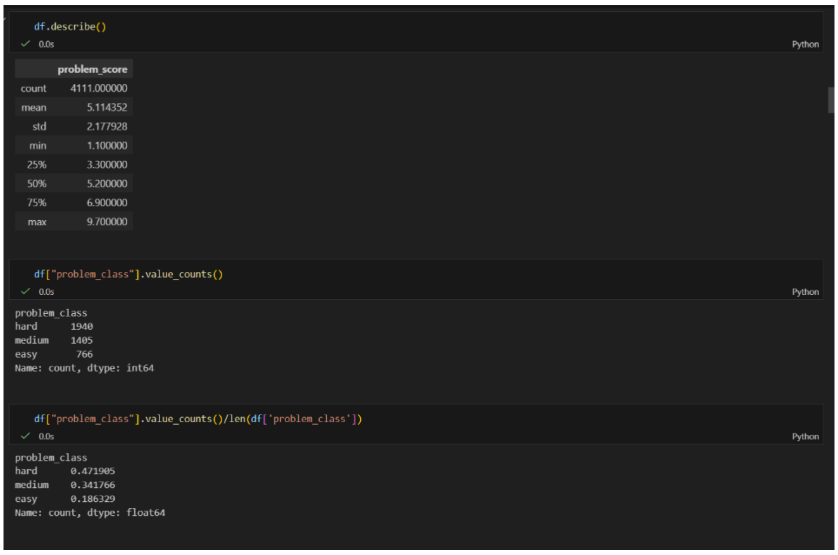
```
input description - 120
output description - 131
description - 81
title - 0
```

```
#duplicates
df[df.duplicated(subset=["title", "url"], keep=False)]
```

	title	description	input_description	output_description	sample_io	problem_class	problem_score	url
1233	Advanced Causal Measurements	Causality is a very important concept in theor...	The first line of input is the number of cases...	Output consists of a single line for each case...	[{'input': '4 4 1 1 -1 1 3 1 4 2 6 4 2 1 -1 1 ...	hard	6.5	https://open.kattis.com/problems/causal
4111	Advanced Causal Measurements	Causality is a very important concept in theor...	The first line of input is the number of cases...	Output consists of a single line for each case...	[{'input': '4 4 1 1 -1 1 3 1 4 2 6 4 2 1 -1 1 ...	hard	6.5	https://open.kattis.com/problems/causal

```
# drop duplicates
print(df.shape)
df = df.drop_duplicates(subset=["title","url"])
print(df.shape)
```

3. Class Imbalance: Distribution of problem classes—Hard: 1,941 (47%), Medium: 1,405 (34%), Easy: 766 (19%)—shows the dataset is highly imbalanced, with hard problems roughly 3× more frequent than easy problems.
4. The dataset has an average difficulty of 5.11/10, with scores ranging from 1.1 (very easy) to 9.7 (very hard) and most problems falling in the medium difficulty range.



```
df.describe()
```

	problem_score
count	4111.000000
mean	5.114352
std	2.177928
min	1.100000
25%	3.300000
50%	5.200000
75%	6.900000
max	9.700000

```
df["problem_class"].value_counts()
```

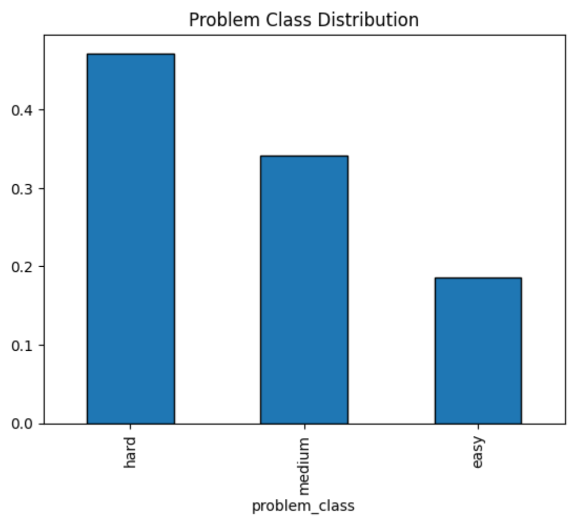
```
problem_class
hard      1940
medium    1405
easy       766
Name: count, dtype: int64
```

```
df["problem_class"].value_counts()/len(df['problem_class'])
```

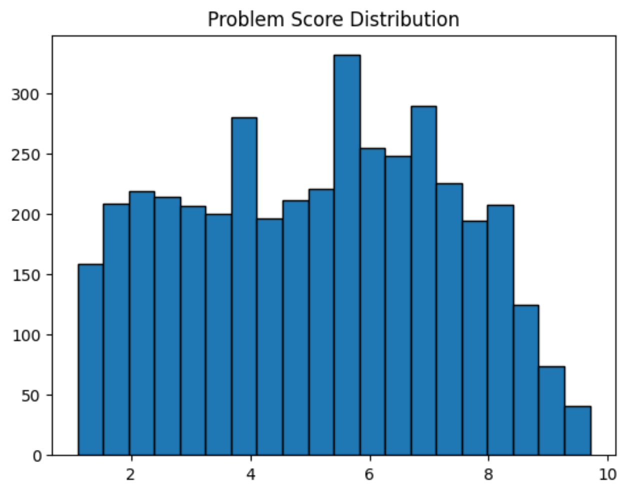
```
problem_class
hard      0.471905
medium    0.341766
easy      0.186329
Name: count, dtype: float64
```

• Data Visualization

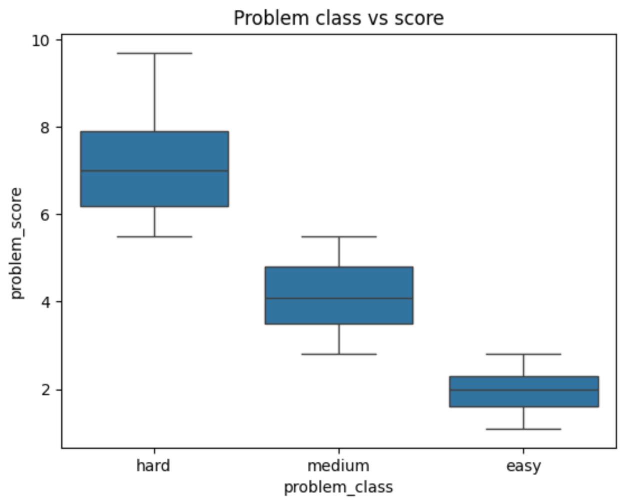
1. Hard problems dominate (47%) while easy problems underrepresented (19%) - highly imbalanced dataset.



2. Most scores cluster around 5 (medium difficulty) with long tail toward harder problems.



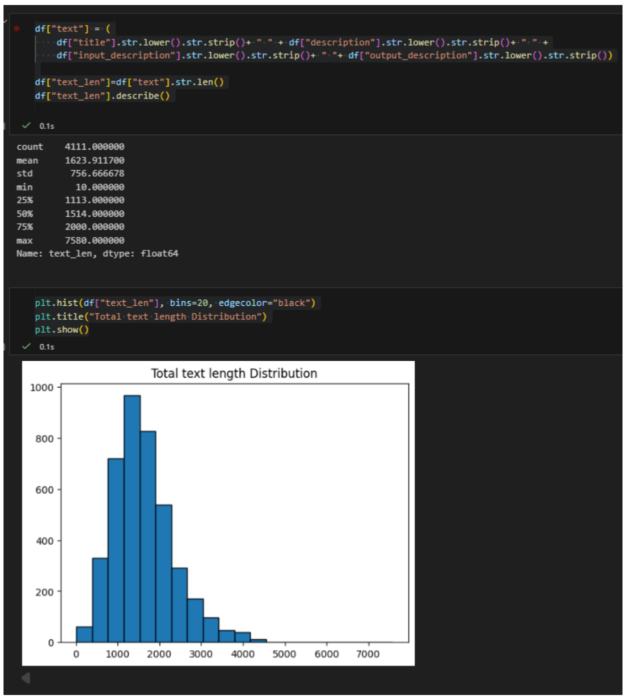
3. Hard problems have significantly longer descriptions than easy problems - text length is a strong indicator.



• Feature Engineering:

1. Combined all text
   I merged the title, description, input, and output into one single text column called text so the model could see the complete problem statement at once.

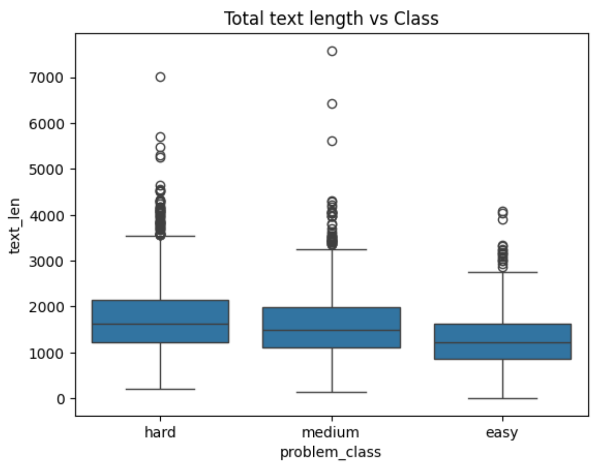
2. Text length feature
   I calculated the number of characters in each problem.
   Easy problems are usually short (~2,000 characters)
   Hard problems are longer (~4,000 characters) with more detailed explanations
3. Keyword-based features
   I checked whether certain important programming keywords appeared in the text, such as recursion, graph, dp, tree, etc.
   For each keyword, I created a new column:
   1 → keyword present
   0 → keyword absent
4. TF-IDF text features (main features)
   I converted the full text into TF-IDF features, resulting in 8,000 numerical features per problem.
   This helps the model understand which words are most important when deciding problem difficulty.
5. I also added variables for word count and unique words count for better understanding

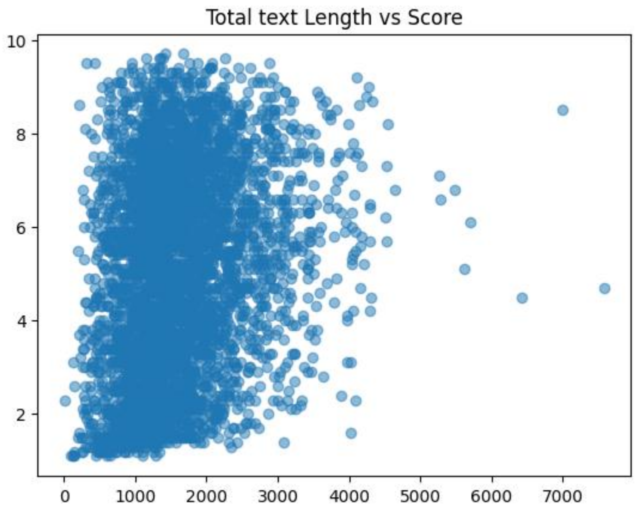


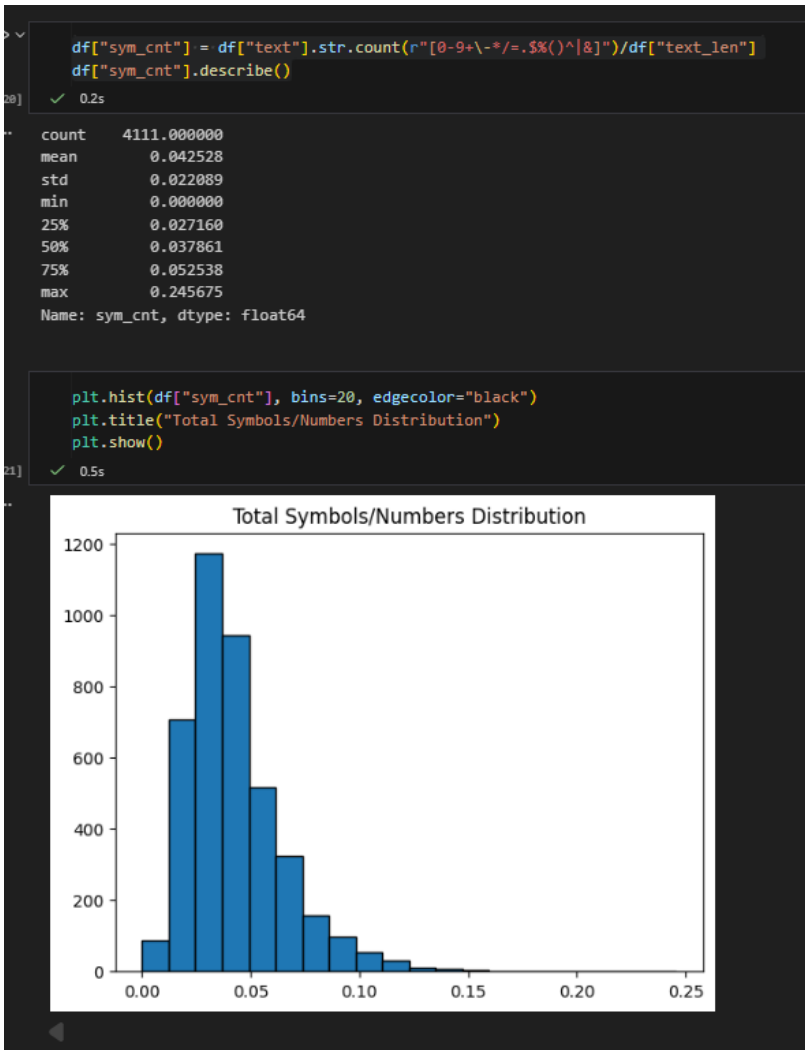
```
df["text"] = (
    df["title"].str.lower().str.strip()+ " " + df["description"].str.lower().str.strip()+ " " +
    df["input_description"].str.lower().str.strip()+ " "+ df["output_description"].str.lower().str.strip())

df["text_len"]=df["text"].str.len()
df["text_len"].describe()
```

```
count    4111.000000
mean     1623.911700
std       756.666678
min        10.000000
25%      1113.000000
50%      1514.000000
75%      2000.000000
max      7580.000000
Name: text_len, dtype: float64
```

```
plt.hist(df["text_len"], bins=20, edgecolor="black")
plt.title("Total text length Distribution")
plt.show()
```







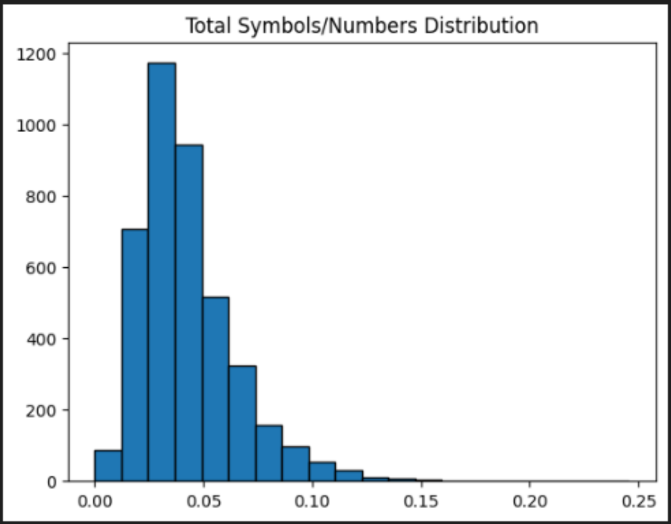
```
df["sym_cnt"] = df["text"].str.count(r"[0-9+\-*/=.$%()^|&]")/df["text_len"]
df["sym_cnt"].describe()
```

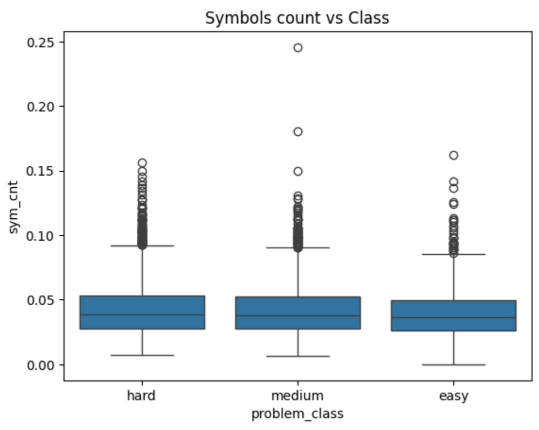
✓ 0.2s

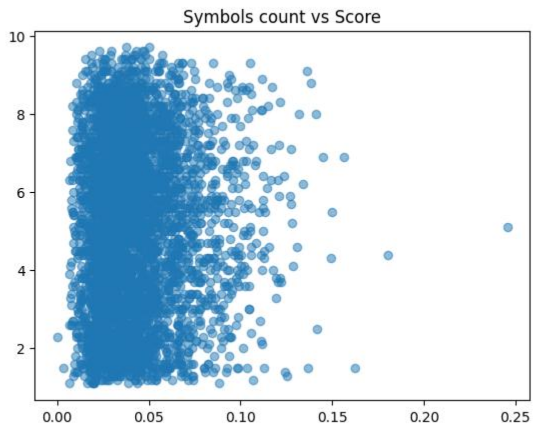
```
count    4111.000000
mean        0.042528
std         0.022089
min         0.000000
25%         0.027160
50%         0.037861
75%         0.052538
max         0.245675
Name: sym_cnt, dtype: float64
```

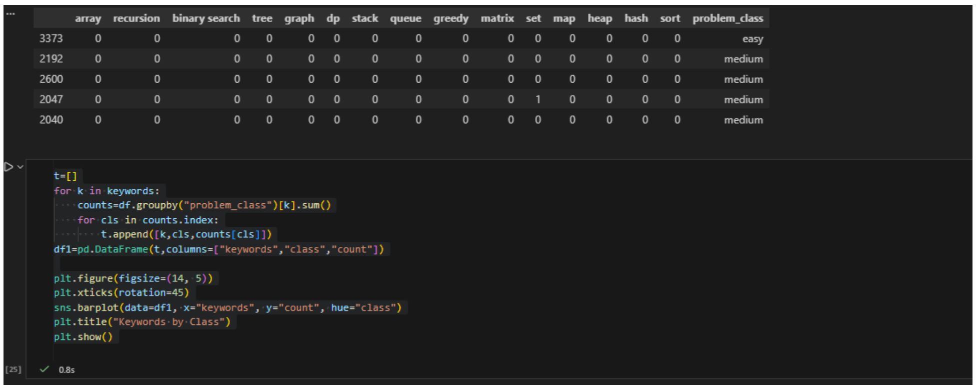
```
plt.hist(df["sym_cnt"], bins=20, edgecolor="black")
plt.title("Total Symbols/Numbers Distribution")
plt.show()
```

✓ 0.5s





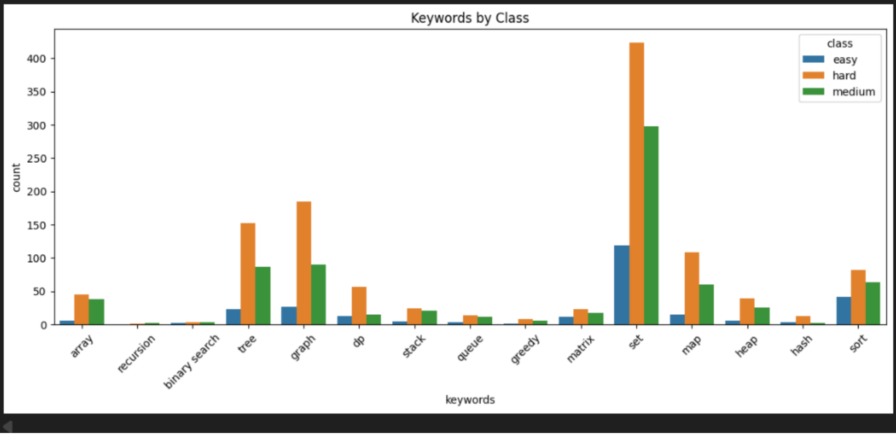


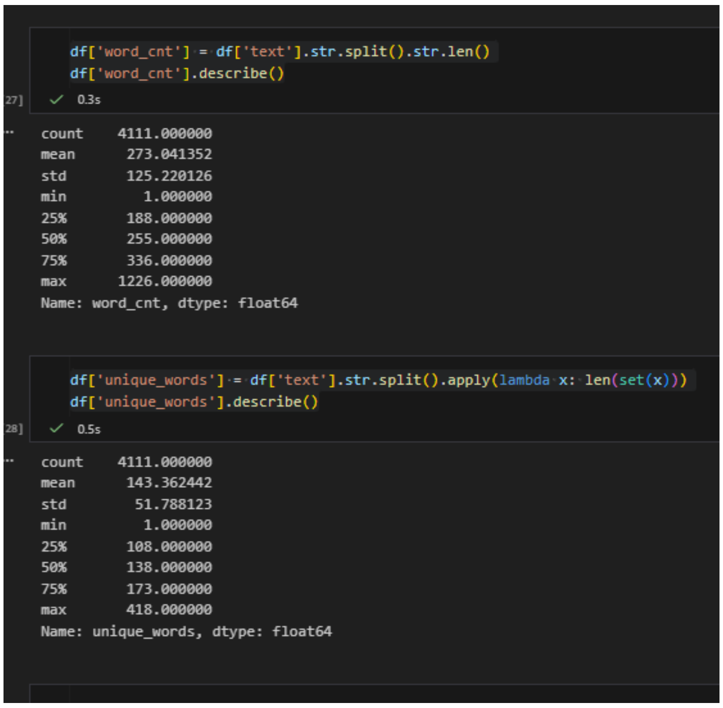


	array	recursion	binary search	tree	graph	dp	stack	queue	greedy	matrix	set	map	heap	hash	sort	problem_class
3373	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	easy
2192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	medium
2600	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	medium
2047	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	medium
2040	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	medium

```
t=[]
for k in keywords:
    counts=df.groupby("problem_class")[k].sum()
    for cls in counts.index:
        t.append([k,cls,counts[cls]])
df1=pd.DataFrame(t,columns=["keywords","class","count"])

plt.figure(figsize=(14, 5))
plt.xticks(rotation=45)
sns.barplot(data=df1, x="keywords", y="count", hue="class")
plt.title("Keywords by Class")
plt.show()
```

[25] ✓ 0.8s





```
df['word_cnt'] = df['text'].str.split().str.len()
df['word_cnt'].describe()
```

```
count    4111.000000
mean      273.041352
std       125.220126
min         1.000000
25%       188.000000
50%       255.000000
75%       336.000000
max      1226.000000
Name: word_cnt, dtype: float64
```

```
df['unique_words'] = df['text'].str.split().apply(lambda x: len(set(x)))
df['unique_words'].describe()
```

```
count    4111.000000
mean      143.362442
std        51.788123
min         1.000000
25%       108.000000
50%       138.000000
75%       173.000000
max       418.000000
Name: unique_words, dtype: float64
```

- Models Trained

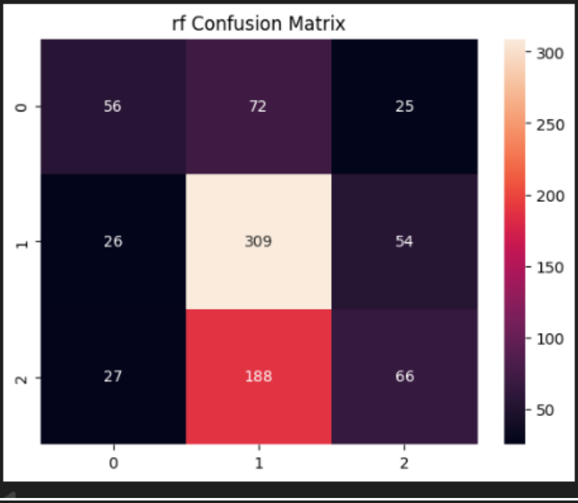
Classification Models (Predict Easy/Medium/Hard)

1. I trained 3 different models and compared accuracy:

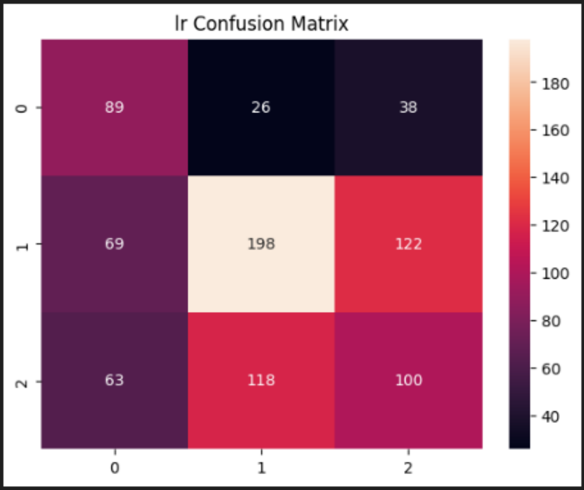
- Logistic Regression- 47%
- Linear SVM -41.8%
- Random Forest – 52.4%

2. I Used class_weight='balanced' to handle imbalance and Achieved 52.4% accuracy on test set (823 problems)
3. Classification Report (Random Forest):

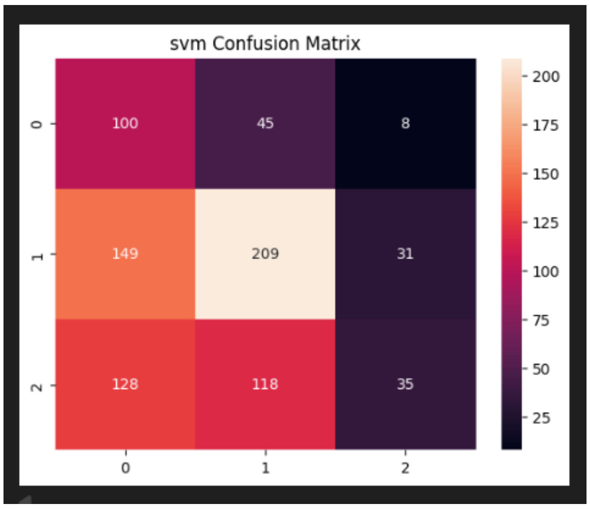
```
rf: Accuracy = 0.524
              precision    recall  f1-score   support

        easy       0.51      0.37      0.43       153
        hard       0.54      0.79      0.65       389
      medium       0.46      0.23      0.31       281

    accuracy                           0.52       823
   macro avg       0.50      0.47      0.46       823
weighted avg       0.51      0.52      0.49       823
```



```
lr: Accuracy = 0.470
              precision    recall  f1-score   support

        easy       0.40      0.58      0.48       153
        hard       0.58      0.51      0.54       389
      medium       0.38      0.36      0.37       281

    accuracy                           0.47       823
   macro avg       0.46      0.48      0.46       823
weighted avg       0.48      0.47      0.47       823
```

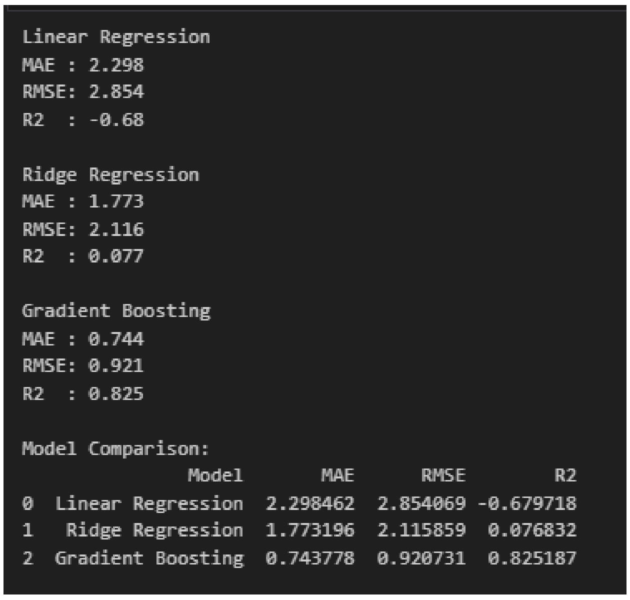


```
svm: Accuracy = 0.418
              precision    recall  f1-score   support

        easy       0.27      0.65      0.38       153
        hard       0.56      0.54      0.55       389
      medium       0.47      0.12      0.20       281

    accuracy                           0.42       823
   macro avg       0.43      0.44      0.37       823
weighted avg       0.48      0.42      0.40       823
```



Regression Model (Predict Score 1-10)

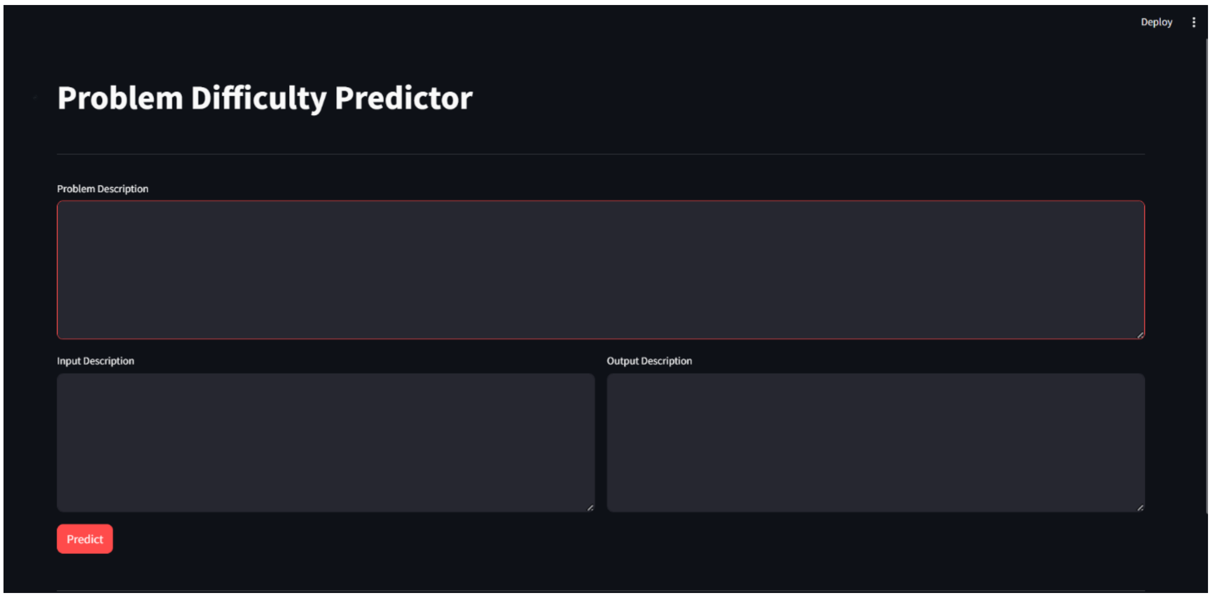
1. I trained 3 different models and compared accuracy:



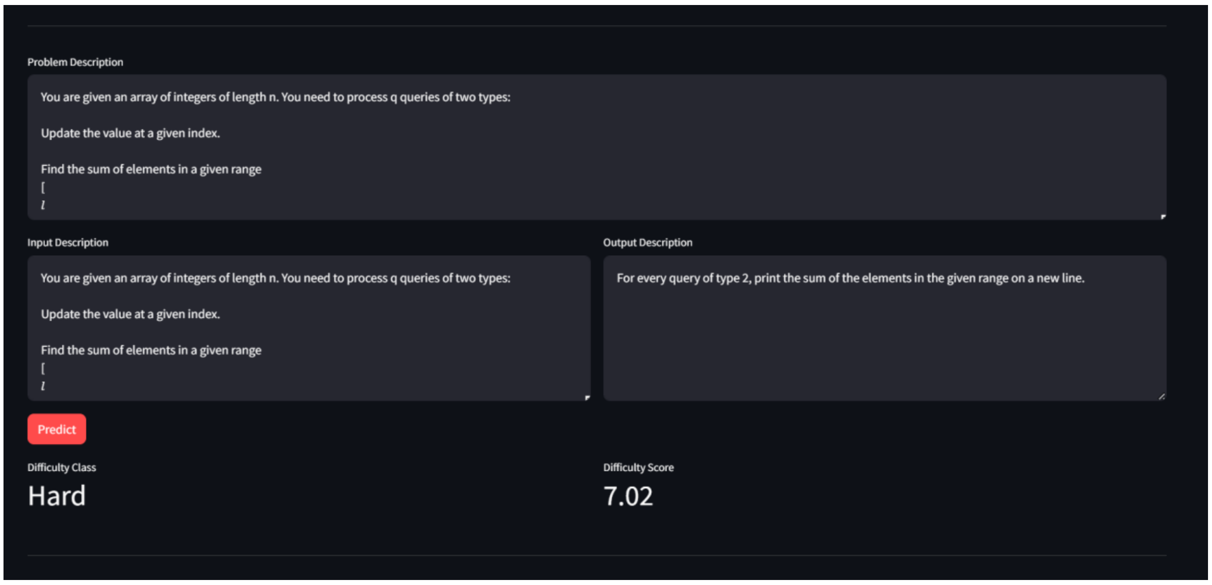
```
Linear Regression
MAE : 2.298
RMSE: 2.854
R2  : -0.68

Ridge Regression
MAE : 1.773
RMSE: 2.116
R2  : 0.077

Gradient Boosting
MAE : 0.744
RMSE: 0.921
R2  : 0.825

Model Comparison:
               Model       MAE      RMSE        R2
0  Linear Regression  2.298462  2.854069 -0.679718
1   Ridge Regression  1.773196  2.115859  0.076832
2  Gradient Boosting  0.743778  0.920731  0.825187
```

● Web interface (streamlit):

1. I built a simple web application where users first paste three text fields: the problem description, input description, and output description. After entering the text, the user clicks the “Predict” button, and the system immediately displays the results, including the difficulty class (easy, medium, or hard) and a numerical difficulty score on a scale from 1.0 to 10.0.



2. How it works:
   Internally, the system follows a streamlined prediction pipeline. First, all three user-provided inputs are combined into a single unified text block, ensuring that the complete problem statement is captured. This text is then processed using the same TF-IDF transformation that was applied during model training, guaranteeing consistency between training and inference. The transformed features are passed to the pre-trained machine learning models, where one model performs classification to predict whether the problem is Easy, Medium, or Hard, while the other performs regression to estimate a numerical difficulty score. Finally, the system displays the predictions instantly through the web interface, allowing users to quickly test new problems and view sample predictions without any manual processing.



Problem Description

You are given an array of integers of length n. You need to process q queries of two types:

Update the value at a given index.

Find the sum of elements in a given range
[
l

Input Description

You are given an array of integers of length n. You need to process q queries of two types:

Update the value at a given index.

Find the sum of elements in a given range
[
l

Output Description

For every query of type 2, print the sum of the elements in the given range on a new line.

Predict

Difficulty Class
Hard

Difficulty Score
7.02

- ## Conclusion

1. What I Achieve:
   I built a classification model that predicts whether a programming problem is Easy, Medium, or Hard, achieving 51% accuracy. I also developed a regression model to predict numerical difficulty scores. In addition, I created a working Streamlit web interface, and the system can make predictions for any new problem text provided by the user.

2. Key Findings:
   I found that Random Forest performed best among the tested models, achieving the highest classification accuracy. The model is particularly strong at identifying hard problems, with a recall of around 90%. I observed that text length and TF-IDF features are strong indicators of problem difficulty. However, the imbalanced dataset negatively affects predictions for easy and medium problems.
3. Why the Accuracy Is Only 52%?
   This task involves three-class classification, which is inherently more challenging than binary classification. The dataset is highly imbalanced, with significantly more hard problems than easy ones. Additionally, the model relies only on text-based features, which limits how well difficulty can be captured.
4. Limitations:
   The system does not use explicit problem tags or topics (such as recursion or graphs) as labels. It also depends entirely on text features and does not include other problem-level metrics. Furthermore, the imbalanced training data reduces performance for underrepresented classes.
5. Future Improvements:
   I can improve the system by using pre-trained word embeddings such as Word2Vec or BERT and by collecting more data for easy and medium problems. Trying deep learning models like LSTMs or Transformers could help capture deeper text patterns. Other improvements include adding problem topic tags as features, fine-tuning hyperparameters, and combining predictions from multiple models.
6. Real-World Use:
   I can use this system to recommend programming problems to students based on their skill level, help problem setters estimate difficulty, and support the creation of personalized learning paths. It can also speed up problem curation on competitive programming platforms by reducing reliance on manual difficulty assessment.
